Escuela Politécnica Superior

Guía de Manejo de Sistemas - ROS, Pepper y Deep Learning

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Jorge Tonda Quesada

Tutor/es:
Miguel A. Cazorla Quevedo

Universitat d'Alacant
Universidad de Alicante

Septiembre 2017

Índice

1. Introducción 3
2. Objetivos 5
3. Estado del arte 6
3.1 Redes neuronales 6
3.2 Caffe deep learning 10
3.3 Faster R-cnn (Towards Real-Time Object Detection with Region Proposal Networks) 10
3.4 Pepper 11
3.5 ROS 12
4. Implementación 14
4.1 Instalación ROS 17
4.2 Crear ROS Workspace 17
4.3 Crear Paquete de Catkin 18
4. 4 Nodos 18
4.5 Tópicos 20
4.6 Ficheros MSG y SRV 24
4.7 Publicador/Suscriptor 25
4.8 ROS_CAFFE – TFG.CPP 26
4. 9 FAST R-CNN 30
5. Conclusión 34
6. Bibliografía y Referencias 35

1. Introducción

Este proyecto de fin de grado está orientado a la rama de robótica y de machine learning. En concreto su objetivo es realizar una guía sobre la puesta en marcha de un sistema de manejo mediante la instalación, configuración y funcionamiento de ROS [1], Pepper [10] [11] y deep learning [9].

Hoy en día, la robótica avanza a un paso por delante del conocimiento. Se diseñan robots casi fuera de las posibilidades humanas, pero uno de los principales problemas actualmente es la capacidad del software para hacer que esos robots sean lo más útiles en el ámbito que han sido construidos. Este problema es una de las barreras más difíciles y que más se ha atragantado (sin grandes avances) estos últimos años en la computación. Esta rama se llama machine learning (dotar de "inteligencia" y "capacidad de aprender" a una computadora o robot).

Existen infinidad de robots que son controlados por personas para realizar sus funciones, como pueden ser robots de rastreo, drones o robots des-activadores de bombas. Este trabajo ocupa una gran cantidad de tiempo que mediante el machine learning se podría llegar a reducir automatizando los procesos mediante aprendizaje.

El machine learning es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadores "aprender". De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

En este proyecto se combinará el machine learning (Caffe) [9] con una herramienta de abstracción en el manejo de robots (ROS), pudiendo así, con la guía, facilitar la creación de robots que detecten objetos, e introducir a cualquier persona que esté interesada en estas tecnologías. El hecho de que un robot

pueda actuar de forma distinta según el código implementado, hace que pueda aplicarse a casi cualquier ámbito, como pueden ser: Los robots de rescate, los robots de reconocimiento, etc... Uno de los más importantes es el ámbito cotidiano.

Comparado con otros ámbitos de la informática, el ámbito relacionado con el manejo de robots y el deep learning puede resultar más complicado a la hora de comenzar un proyecto desde 0. Dada la gran cantidad de robots que utilizan este tipo de tecnología (sigue incrementándose actualmente) y la dificultad añadida al inicio, existe una barrera de conocimiento que impide o echa para atrás a gran parte de las personas que intenta realizar un proyecto en esta rama. Por lo que, mediante una guía sobre la puesta en marcha de un sistema de manejo (instalación, configuración y funcionamiento de ROS, Pepper y deep learning) se podría reducir esta barrera de conocimiento.

Para abordar el tema podemos ver un ejemplo claro de la problemática actual.

Una persona tiene una idea y decide realizar un proyecto mediante deep learning. Esta persona tiene relativamente pocos conocimientos en el ámbito de la informática, por lo que le es difícil familiarizarse con las tecnologías, y lo que es más importante, qué tecnologías utilizar y dónde obtener la información adecuada. Mediante esta guía de adaptación y puesta en marcha de un sistema, será capaz de llevar a cabo su proyecto de forma fácil.

El proyecto tratará en primer lugar la herramienta ROS (Robot Operating System), para ayudar a los desarrolladores de software a crear aplicaciones robóticas, mediante su instalación, configuración, puesta a punto y creación de paquetes con Catkin. Uno de los puntos más importantes es la utilización de Redes neuronales de aprendizaje, Caffe (machine learning), con la creación de dichos paquetes de Catkin. Por otra parte tratará la unión de ROS con el robot Pepper mediante el simulador de Gazebo (tópicos, nodos, paquetes, publicadores...). Con un robot de la clase Pepper se consumirá esta red

neuronal, Caffe, y se obtendrá el tipo de objeto mostrado a través de la cámara frontal del Pepper. Por último se presentará una red neuronal que es capaz de obtener ciertas neuronas activas para saber en qué parte de la imagen está el objeto detectado, como es Faster R-cnn (Red neuronal paralela a caffe en python).

2. Objetivos

El objetivo principal de este proyecto final de Grado es crear una guía para el manejo de sistemas basada en ROS, Pepper y deep learning. Este, está dividido en 3 sub objetivos.

El primer objetivo es poner en marcha el simulador Gazebo [8] del robot Pepper, al mismo tiempo que se pone en funcionamiento el sistema operativo ROS para realizar la conexión con dicho robot Pepper.

El segundo objetivo es conseguir que el robot reconozca y distinga objetos con la ayuda de Caffe deep learning. Esto se hará mediante ROS Jade y su sistema de macros e infraestructura, Catkin.

Para finalizar, se profundiza en Faster R-Cnn, una red neuronal convolucional derivada de Caffe, que puede obtener las coordenadas del objeto reconocido.

3. Estado del arte

3.1 Redes neuronales

Aunque existen varias maneras de implementar *Deep Learning* [3], una de las más comunes es utilizar redes neuronales [7]. Una red neuronal es una herramienta matemática que modela, de forma muy simplificada, el funcionamiento de las neuronas en el cerebro. Es una serie de operaciones matemáticas que a partir de un conjunto de datos obtiene otro conjunto de datos tratados.

Figura 1 Neurona Biológica

En concreto y como ejemplo, nosotros vamos a tratar una red neuronal que detecte objetos en imágenes (Faster R-cnn y Caffe). Para crear los datos de entrada, hay que codificar en forma de números los píxeles de la imagen a tratar. Si la imagen es a color, cada píxel representará 3 números y si no lo es, solo representará 1. La salida es un conjunto de números, siendo el más próximo a 1.0 el que de verdad indique la naturaleza del objeto (Si gato es el cuarto número de la salida y perro el tercero y está más cercano al 1 el tercer número, la imagen corresponderá a un perro). Si toma un valor cercano a 0.0 significa que no lo hay. Valores intermedios se pueden interpretar como inseguridad, o probabilidad.

En la Faster R-Cnn una característica especial es saber dónde se encuentra el objeto o animal en la imagen utilizando una recolección de las neuronas que se han activado en el proceso, indicando en que coordenadas de la imagen se encuentra el objeto. Esto lo explicaremos mejor cuando veamos la Faster R-Cnn.

3.1.1 Arquitectura

La siguiente figura nos va a mostrar un ejemplo de cómo está constituido una sola neurona.

- Un conjunto de entradas $x1,\ldots xn$
- Los pesos sinápticos $w1,\ldots wn$ correspondientes a cada entrada.
- Una función de agregación, Σ
- Una función de activación, f
- Una salida, Y.

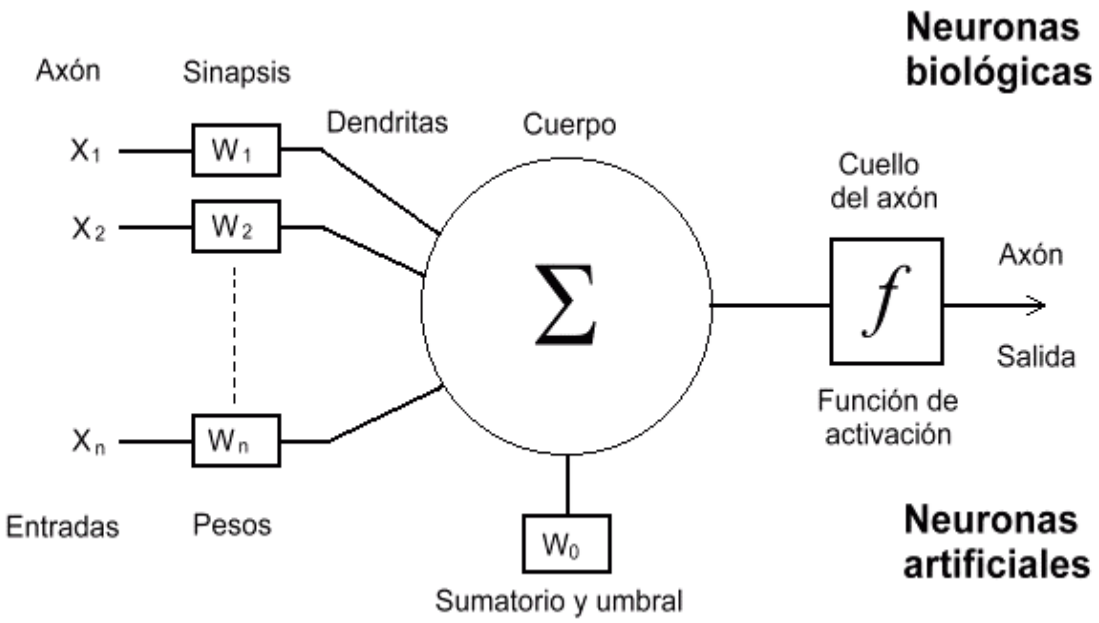


Figura 2 - Esquema neurona artificial

En el siguiente diagrama podemos ver la arquitectura de una red neuronal básica. Cada círculo representa una neurona. Las neuronas se organizan en capas, de la siguiente forma: las neuronas **amarillas** son las entradas, y reciben cada uno de los números de nuestra lista de números entrante, las neuronas **verdes** son las salidas, y una vez que la red realiza su operación matemática, contienen el resultado, también como una lista de números; las neuronas **grises** son neuronas ocultas, que contienen cálculos intermedios de la red. Debajo de la imagen queda mejor explicado.

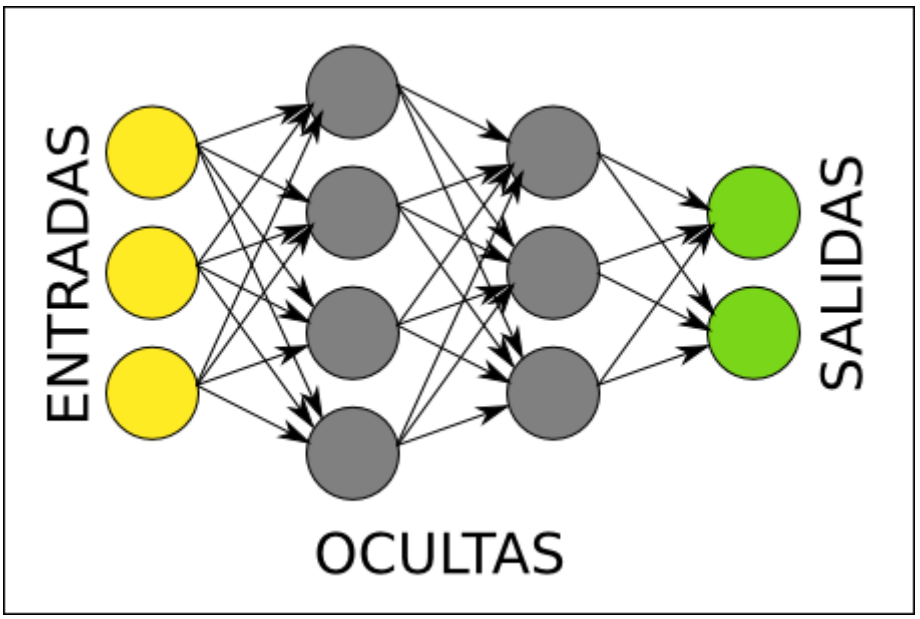


Figura 3 - Ejemplo red neuronal básica

Neuronas de entrada: Aquellas que reciben los estímulos o la información externa que proviene directamente del emisor, obteniendo así los valores de entrada. Esta información se transmite a neuronas internas que se ocupan de su procesamiento.

Neuronas ocultas: Pueden existir una o varias capas. En este segundo nivel se produce cualquier tipo de representación interna. No tienen ningún contacto con la información de entrada ni con la salida.

Neuronas de Salida: Encargadas de recoger la información procesada por las

neuronas ocultas y proporcionar la respuesta al sistema.

3.1.2 BackPropagation

El algoritmo utilizado para calcular las salidas es llamado **Propagación hacia atrás** (Backpropagation).

En una red Backpropagation existe una capa de entrada con N neuronas y una capa de salida con M neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida). No hay conexiones hacia atrás feedback ni laterales entre las neuronas de la misma capa. La aplicación del algoritmo tiene dos fases, una hacia delante y otra hacia atrás.

Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red, se inicia la segunda fase, comparándose éstos valores con la salida esperada para así obtener el error. Se ajustan los pesos de la última capa proporcionalmente al error. Se pasa a la capa anterior con una retropopagación del error, ajustando los pesos y continuando con este proceso hasta llegar a la primera capa. De esta manera se han modificado los pesos de las conexiones de la red para cada patrón de aprendizaje del problema, del que conocíamos su valor de entrada y la salida deseada que debería generar la red ante dicho patrón. La técnica Backpropagation requiere el uso de neuronas cuya función de activación sea continua, y por lo tanto, diferenciable. Generalmente, la función utilizada será del tipo sigmoidal.

La importancia de la red backpropagation consiste en su capacidad de **auto adaptar los pesos** de las neuronas de las capas intermedias para aprender la relación que existe ente un conjunto de patrones de entrada y sus salidas

correspondientes. La red debe encontrar una representación interna que le permita generar las salidas deseadas cuando se le dan entradas de entrenamiento, y que pueda aplicar, además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas.

3.2 Caffe deep learning

Caffe [11] [12] es un framework de aprendizaje profundo (deep learning) que funciona con expresiones, velocidad y modularidad. Fue desarrollado por Berkeley Al research y contribuidores de distintas plataformas.

Caffe es capaz de cambiar entre CPU y GPU para entrenar los distintos conjuntos de neuronas. Es capaz de trabajar a una velocidad de 4 imagenes/ms para entrenar conuntos. Para este proyecto hemos elegido este framework porque dispone de una extensa documentación y una gran comunidad de desarrolladores que permite obtener un feedback de fallos y problemas comunes facilitando el desarrollo.

En concreto en esta guía utilizamos Caffe con ROS y entramos levemente en la red neuronal derivada de Caffe, Faster R-cnn, explicada a continuación.

3.3 Faster R-cnn (Towards Real-Time Object Detection with Region Proposal Networks)

La R-cnn [4] [5] [6] es un framework de detección de objetos basado en redes convolucionales profundas, las cuales incluyen una Region Proposal Network (RPN) y una red de detección de objetos. Las dos redes están previamente entrenadas para compartir las capas convolucionales para testear rápidamente.

Las capas convolucionales tiene el efecto de filtrar la imagen de entrada con un

núcleo previamente entrenado. Esto transforma los datos de tal manera que ciertas características (determinadas por la forma del núcleo) se vuelven más dominantes en la imagen de salida al tener estas un valor numérico más alto asignados a los píxeles que las representan. Estos núcleos tienen habilidades de procesamiento de imágenes específicas, como por ejemplo en nuestro caso la detección de las zonas de una imagen que pueden contener un objeto.

3.4 Pepper

Pepper es un robot humanoide capaz de interactuar con las personas. Su tecnología permite detectar tanto el lenguaje verbal como el no verbal. Es capaz de reconocer la posición de la cabeza y el tono de la voz para reconocer el estado emocional e actuar en consecuencia. Dispone de una cámara 3D y 4 micrófonos que le permite comunicarse de forma fluida. Respecto a sus movimientos, son flexibles e imitan los gestos humanos en lo posible. Puede desplazarse en cualquier dirección hasta 3 km/h y es capaz de mover sus extremidades o cabeza de forma independiente.

Pepper permite una programación en distintos lenguajes, facilitando así un amplio espectro de potenciales usuarios. Estos lenguajes son Choreographe, Python y C++. Es una herramienta perfecta tanto para iniciación en la programación como para desarrollo e investigación de nuevas aplicaciones.

Información técnica

- **Peso**: 28 kg
- **Altura**: 120 cm.
- **Fondo**: 42,5 cm.
- **Batería**: Litio, 30,0Ah/795Wh
- **Autonomía**: Hasta 12 horas.

- **Conectividad**: Wi-Fi / Ethernet
- **Velocidad**: Más de 3km/h
- **Motores**: 20
- **Partes móviles**: Cabeza (1), hombros (2), codos (2), muñecas (2), dedos (10), caderas (1) y rodillas (1)
- **Ruedas**: 3 (omnidireccionales)
- **Movimiento**: 360º
- **Velocidad máxima**: 3 km/h
- **Tablet**: LG CNS

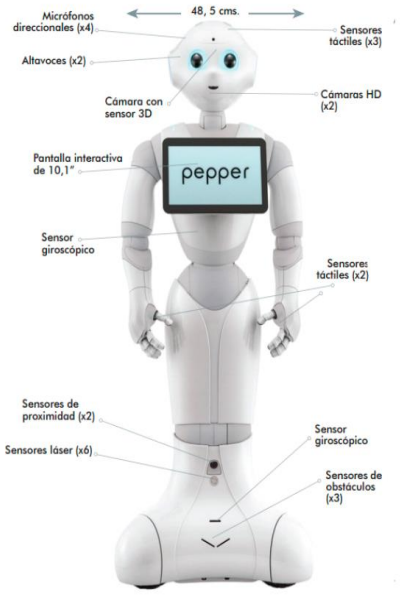


Figura 4 - Esquema Robot Pepper

3.5 ROS

ROS (Robot Operating System) proporciona bibliotecas y herramientas para ayudar a los desarrolladores de software crear aplicaciones robóticas. Proporciona una abstracción del hardware, de los controladores de dispositivos, las bibliotecas, visualizadores, paso de mensajes, gestión de paquetes y mucho

más.

Este conjunto de herramientas y librerías nos proporciona:

- **Mecanismo de comunicación entre programas:**

Es un estándar para comunicar entre sí diferentes programas de un mismo sistema, ya sea en el mismo computador o en varios computadores. La computación distribuida es un recurso muy común donde pequeñas partes de un robot trabajan por separado para conseguir un objetivo común.

- **Reusabilidad de código:**

Los paquetes estándar proporcionados en las distribuciones de ROS implementan muchos de los algoritmos comúnmente usado en robótica que ya han sido depurados y usados de forma estable. Además, el modelo de comunicación y gestión de mensajes de ROS se ha convertido en estándar y muchas de las plataformas robóticas ya implementan interfaces para ser usadas directamente desde ROS. Además de los paquetes estándar, existen muchos otros en la comunidad que implementan interfaces para sus sistemas (robots, sensores, librerías...)

- **Testeado rápido:**

Debido al diseño de comunicación por paso de mensajes, ROS nos permite simular muchos de los dispositivos con los que nuestro sistema trabajará de forma que podemos aislar la funcionalidad de nuestro sistema del código de comunicación entre las diferentes partes del sistema, sensores y actuadores. Uno de los aspectos claves al desarrollar una aplicación es la capacidad de repetir los experimentos y poder simular los sensores y actuadores que nos permite crear conjuntos de prueba.

A continuación, vamos a explicar los conceptos clave de ROS:

- **Tópico** (topic): es una canal de información entre los nodos. Un nodo puede

suscribirse o publicar en un tópico. Por ejemplo, stage (simulador de robots), emite un tópico /odom que es la odometría del robot.

- **Paquete** (package): es la forma en la que está organizado ROS. Puede con tener un nodo, un conjunto de datos o lo necesario para construir un módulo e implementar distintas funcionalidades para tareas de localización, navegación, mapeado, simulación, etc.

- Nodo (node): es un proceso ejecutable que puede realizar varias acciones, como controlar el láser, otro los motores y otro la construcción de mapas.

- Pila (stack): es un conjunto de nodos que proporciona alguna funcionalidad (por ejemplo, la pila de navegación).

4. Implementación

Para realizar la implementación hemos seguido un patrón de diseño. El objetivo de este patrón es crear una Pila de nodos que sean capaces de realizar todas las funciones de nuestro trabajo (Pila de reconocimiento). Todo estará almacenado en un paquete de ROS. Este esquema está basado en un arquitectura publicación/suscripción de tópicos entre nodos. El esquema a seguir es el siguiente:

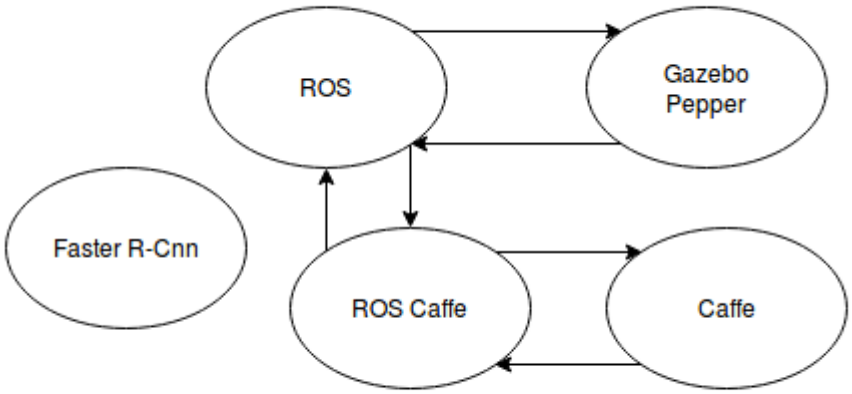
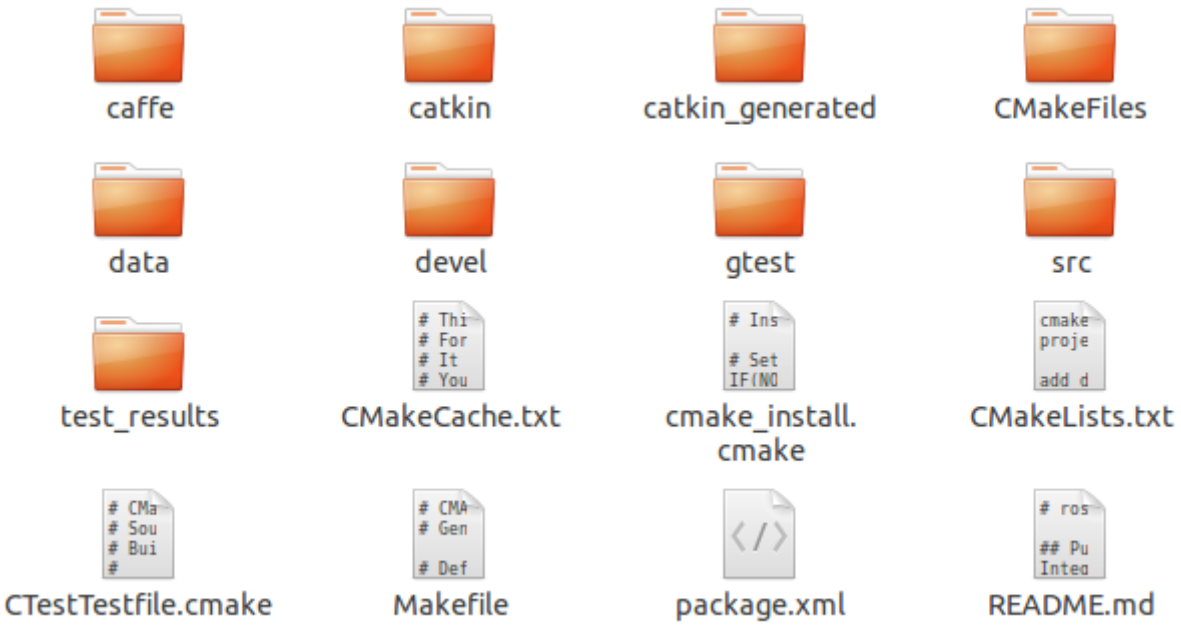


Figura 5 - Esquema implementación

El Ros caffe es un paquete de ROS que adapta la utilización de caffe para ROS. Este nodo se comunica con Caffe cuando recibe las imagenes del tópico de Gazebo y devuelve una serie de Tags (strings) con la respuesta procesada. A parte de realizar está implementación, se presenta la red Faster R-Cnn.

En el proyecto hemos creado un paquete ROS_Caffe con esta estructura.



Dentro del directorio Src, hemos creado un fichero llamado tfg.cpp, dónde se encuentra el código del proyecto. Dicho código está explicado más abajo en Publicador/suscriptor. Para ejecutar dicho código debemos realizar los pasos de compilación de Catkin, iniciando primero roscore.

```
$ roscore
```

```
$ catkin_make
$ source devel/setup.bash
```

Una vez el código está compilado, se ejecuta el simulador del robot Pepper Gazebo mediante el comando:

```
$roslaunch pepper_gazebo_plugin pepper_gazebo_plugin_Y20.launch
```

Iniciamos el simulador y utilizamos el siguiente comando para ejecutar el nodo tfg.cpp

```
$ rosrun ros_caffe tfg
```

Para visualizar la salida, debemos visualizar el tópico "/res_caffe" mediante esta linea de código.

```
$ rostopic echo /res_caffe
```

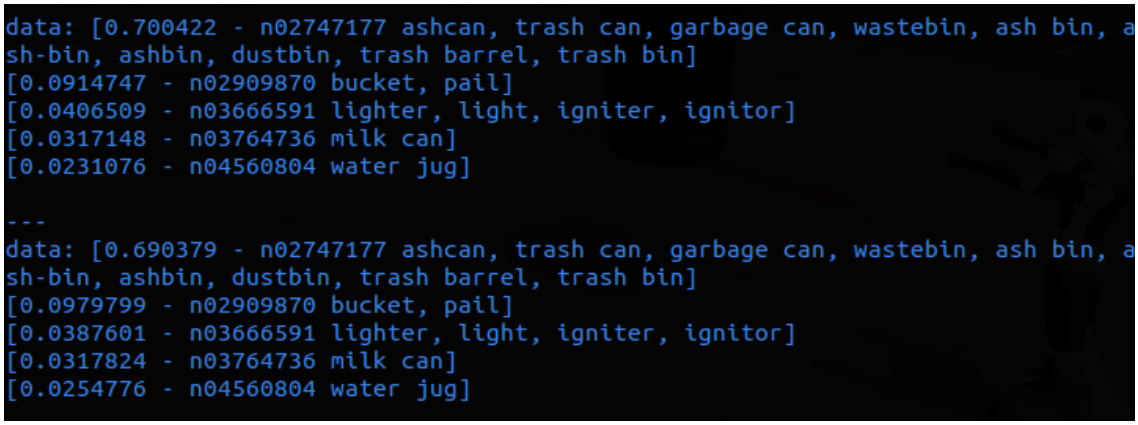


Figura 6 - Salida Tópico res_caffe

Una vez descrito todo el patrón de diseño y las tecnologías que vamos a utilizar, empezamos con la Guía de Instalación y puesta en marcha tanto de ROS como de todos sus componentes.

4.1 Instalación ROS

Para comenzar instalamos ROS y configuramos el entorno. Ros se puede organizar en Rosbuild o Catkin. En este caso vamos a utilizar Catkin ya que es más recomendable para organizar código, debido a la utilización de Cmake y su flexibilidad.

4.2 Crear ROS Workspace

Una vez está instalado ROS, creamos el directorio donde vamos a inicializar el Workspace de ROS.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

El comando Catkin_make lo utilizamos para inicializar el workspace. Con este comando en el directorio principal se crean las carpetas 'build' y 'devel', además, se crea un CmakeLists.txt en el directorio 'src'. Para finalizar con la configuración del Workspace se ejecuta este comando.

```
$ source devel/setup.bash
```

4.3 Crear Paquete de Catkin

Creamos un paquete con las dependencias necesarias para realizar las acciones de paso de imágenes y procesamiento de las mismas.

```
$ catkin_create_pkg reconocimiento-objetos cv_bridge genmsg
image_transport message_generation roscpp roslib sensor_msgs
std_msgs
```

Se crea un paquete de Catkin con el nombre indicado en el primer parámetro “reconocimiento-objetos” y un CmakeList.txt con todas las dependencias indicadas a partir de ese parámetro.

Una vez realizado, construimos los paquetes de catkin en el workspace y

```
$ catkin_make
$ source ./devel/setup.bash
```

ejecutamos el archivo de configuración generado en el apartado anterior.

4. 4 Nodos

Para comenzar con los nodos, primero tenemos que ejecutar en un terminal el Roscore.

```
$ roscore
```

Una vez está inicializado puedes ver todos tus nodos con el comando:

```
$ rosnode list
```

Con roslaunch ejecutamos un paquete. Estos nodos que se crean los podremos ver con el comando rosnode.

```
 $ roslaunch pepper_gazebo_plugin
pepper gazebo plugin Y20.launch
```

Se abrirá una ventana donde se simula todos los nodos y todas las posibles piezas (movimientos) de un robot Pepper. En este simulador puedes acceder y variar todos los datos de las diferentes partes de la simulación, como pueden ser partes del robot o partes del terreno. Tendrá un aspecto similar a este.

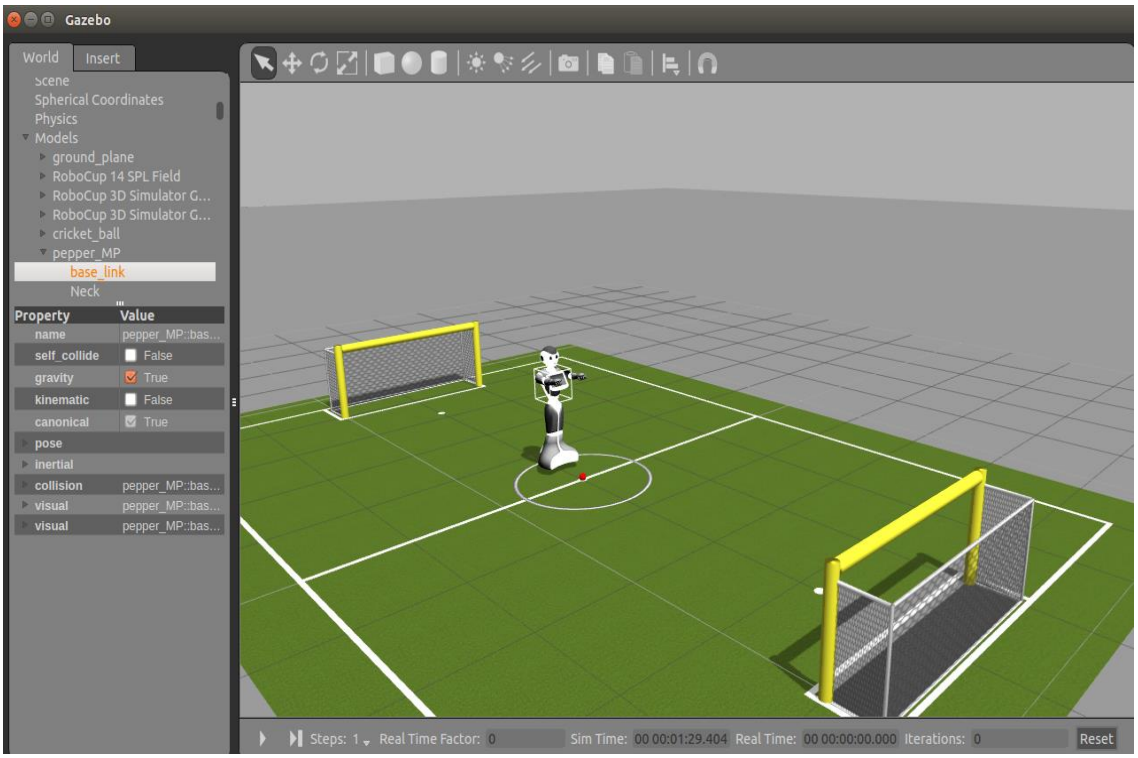


Figura 7 - Simulador Gazebo

El resultado de ejecutar el comando "rosnode list", se puede apreciar que aparecen los nuevos nodos: Peppr_trajectory_controller y Gazebo.

```
tonda@TondaPC:~$ rosnode list
/rosout
tonda@TondaPC:~$ rosnode list
/gazebo
/pepper_trajectory_controller
/rosout
/turtlesim
```

4.5 Tópicos

Los nodos se comunican entre ellos mediante los tópicos. Estos nodos publican o se suscriben mediante dichos tópicos. Por ejemplo, si un nodo de Gazebo comparte información, lo hará publicándola en un tópico el cual será suscrito

por otro nodo. Para visualizar los tópicos que conectan con los nodos utilizamos rqt_graph. (Este comando solo funciona correctamente con la distribución indigo de ROS. En nuestro caso tenemos Jade).

```
$ rosrun rqt_graph rqt_graph
```

Podemos listar los tópicos y obtener todo tipo de información de ellos con algunos comandos como bw, echo, hz, find, list, info, pub, type. Todos estos precedidos por rostopic. Cada uno de ellos muestra información diferente.

```
 $ rostopic
Echo bw hz find list info pub type
```

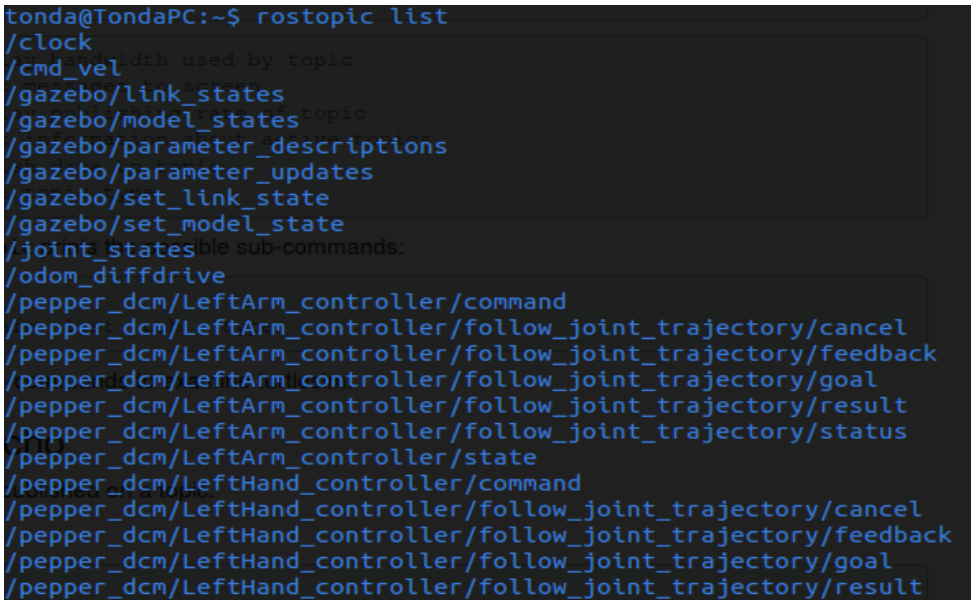


Figura 8 - Lista de tópicos disponibles

Cada tópico muestra una información diferente del robot. Con rostopic type obtenemos el tipo de dato que emite y con rosmsg show, podemos obtener detalles sobre esa salida (Número de datos que se envían y el tipo de esos datos). En nuestro caso nos interesa obtener la salida de la cámara frontal para poder utilizarla con Fast-RCNN.

```
$ rostopic type pepper_robot/camera/front/image_raw
$ rosmsg show sensor_msgs/Image
```

Figura 9 - Tipo de datos del tópico Camera

Si queremos publicar cualquier dato en un tópico utilizamos "pub".

```
$ rostopic pub [topic] [msg_type] [args]
```

Para ver los logs y los mensajes de forma más visual se utiliza rqt_console y rqt_logger_level. Esto nos puede ayudar a ver que mensajes se están emitiendo y de qué tipo son.

```
$ rosrun rqt_console rqt_console
$ rosrun rqt_logger_level rqt_logger_level
```

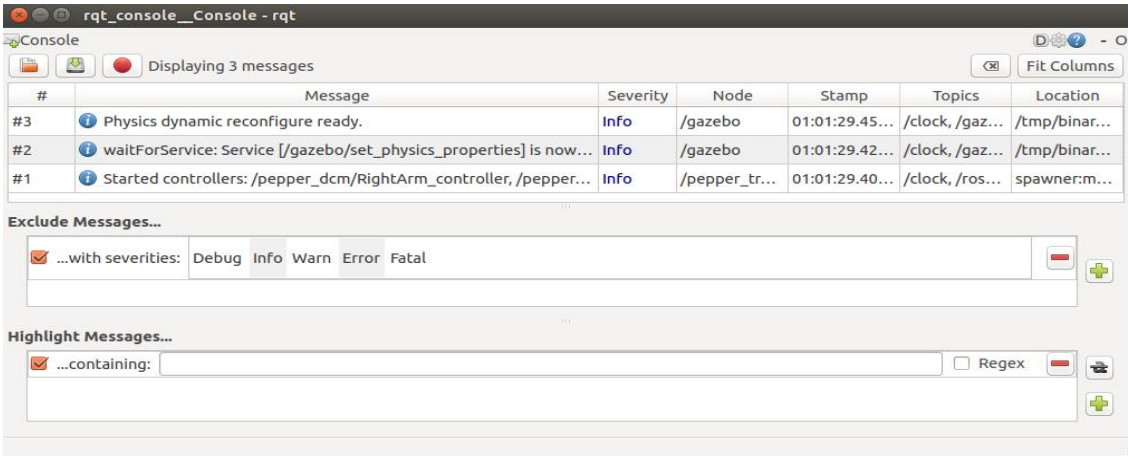


Figura 10 - Rqt_console mensajes de información

Con el gazebo inicializado, podemos observar que se están mostrando 3 mensajes de información. El primero de ellos muestra un mensaje sobre las físicas, el segundo sobre las propiedades de las mismas y el tercero indica que los controladores de las extremidades y manos del robot están iniciados.

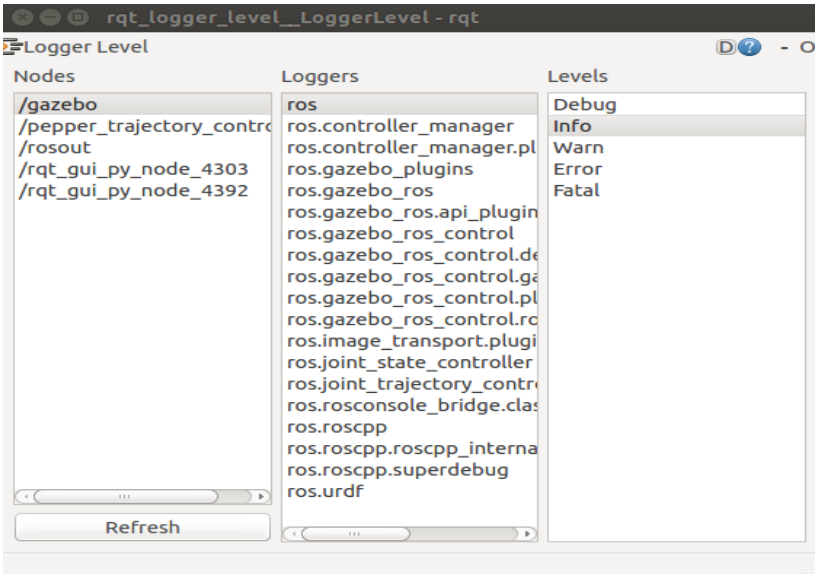


Figura 11 - Logers existentes

En la ventana rqt_logger podemos observar los loggers que existen y los niveles de información que queremos recibir del mismo.

4.6 Ficheros MSG y SRV

Estos tipos de fichero se utilizan para indicar el tipo de salida de los nodos. En nuestro proyecto utilizamos un tipo msg creado directamente en el código.

Msg: Son ficheros simples de texto que describen los campos de un mensaje de ROS. Son utilizados para fijar la salida de código para diferentes lenguajes.

Srv: Son ficheros que describen un servicio. Están compuestos por dos partes: El envío y la respuesta, separados por una linea "---".

Los tipos simples que pueden utilizar los msgs, además de los tipos compuestos (Como 'sensor_msgs/Image image') son:

- int8, int16, int32, int64
- float32, float64
- string
- time, duration
- Otro archivo msg
- array[] de tamaño variable y array[C] de tamaño fijo

El fichero salida.msg contendrá estas lineas de código, las cuales indican que la salida será del tipo sensor_msgs/Image y del tipo String.

```
sensor_msgs/Image image
string tag
```

Por otro lado, para volcar el fichero en código C++ (o otro lenguaje) tenemos que descomentar las lineas siguientes del fichero "package.xml".

```
<build_depend>message_generation</build_depend>
 <run_depend>message_runtime</run_depend>
```

```
find_package(catkin REQUIRED COMPONENTS
  cv_bridge
 ...
  roslib
  sensor_msgs
  std_msgs
 message_generation)
```

Una vez realizados estos pasos, incluimos las dependencias del paquete y el/los archivo/s en el fichero CmakeList.txt

```
add_message_files(
  FICHEROS
  salida.msg)
```

Respecto a los ficheros Srv, se utilizan de la misma forma, con la diferencia de que tienen un envío y una respuesta. Los tipos que se reciben o se envía se separan con una linea de guiones "---". Quedaría de esta manera.

```
sensor_msgs/Image image
---
string tag
```

4.7 Publicador/Suscriptor

Un Nodo es un ejecutable que está conectado a la red de ROS. En nuestro proyecto vamos a crear un suscriptor, que consuma el tópico de la cámara frontal del Pepper para, a su vez, analizar con caffe la imagen y publicar en un

tópico los resultados de dicho análisis.

Este nodo se llama TFG.CPP y está ubicado en el paquete ROS_Caffe.

4.8 ROS_CAFFE – TFG.CPP

El nodo ejecutable tfg.cpp se suscribe al tópico "pepper_robot/camera/front/image_raw" enviado por el simulador de gazebo. Las imagenes que recibe de dicho tópico son procesadas por Caffe y envía los resultados publicándolos en un tópico, llamado "/caffe_ret". A continuación se puede ver una aproximación de dos casos prácticos de como se vería la puesta en marcha y el funcionamiento de caffe.

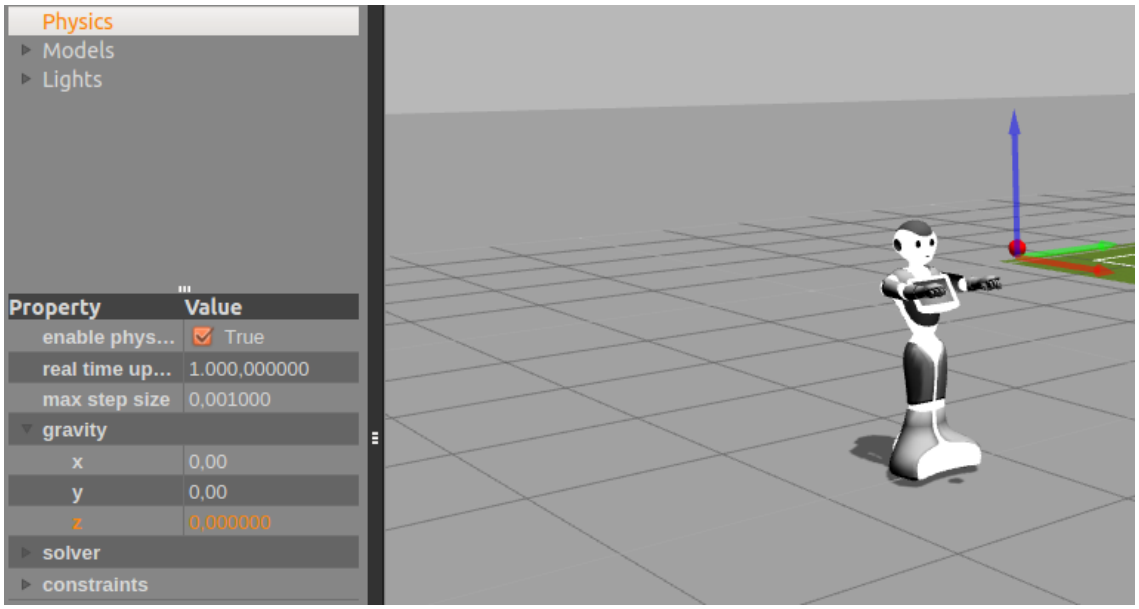


Figura 12 - Físicas del simulador

Como podemos observar, cambiamos el valor de las físicas para que no actúen en los objetos donde los colocamos.

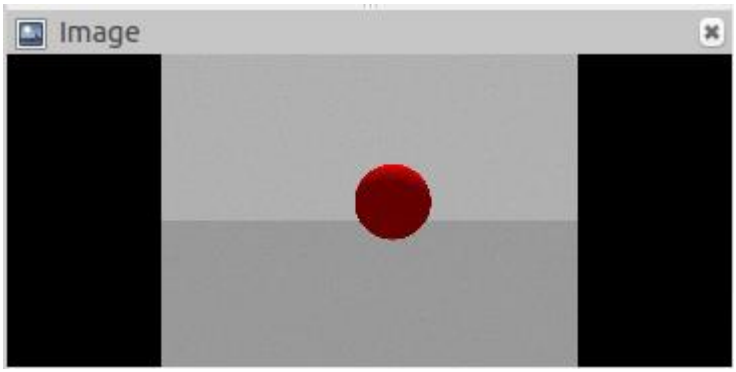


Figura 14 - Visión Camara Frontal pelota

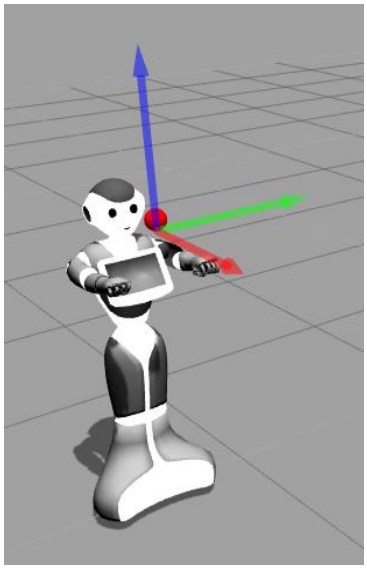


Figura 13 - Robot Pepper frente a objeto pelota

La imagen de la izquierda, muestra la imagen enviada por el simulador de Pepper Gazebo al tópico "pepper_robot/camera/front/image_raw" y recibida por el nodo en el mismo.

```
data: [0.780793 - n02782093 balloon]
[0.120012 - n03942813 ping-pong ball]
[0.0431221 - n04116512 rubber eraser, rubber, pencil eraser]
[0.0095578 - n03929660 pick, plectrum, plectron]
[0.00585365 - n03729826 matchstick]

---
data: [0.874679 - n02782093 balloon]
[0.0746706 - n03942813 ping-pong ball]
[0.0125192 - n04116512 rubber eraser, rubber, pencil eraser]
[0.00845712 - n03929660 pick, plectrum, plectron]
[0.00535152 - n03888257 parachute, chute]

---
data: [0.823219 - n02782093 balloon]
[0.089254 - n03942813 ping-pong ball]
[0.0258065 - n04116512 rubber eraser, rubber, pencil eraser]
[0.0140148 - n03929660 pick, plectrum, plectron]
[0.00616055 - n09229709 bubble]
```

Figura 15 - Resultado Caffe con Ballon

Como podemos comprobar, el tópico de salida "/caffe_res", muestra una serie

de números, que van del 0 al 1 (cuanto más cercano al 1 más se parecerá al objeto marcado). En la imagen superior se han analizado 3 imágenes de la pelota roja. En este caso la pelota se iba moviendo ligeramente hacia arriba, por lo que, los resultados varían mínimamente los unos de los otros. Esta variación se puede apreciar en los números de la parte superior de cada paquete de envío su proximidad al 1. Esto, unido a que las demás estimaciones están cerca del 0, indica que el objeto probablemente sea una Pelota (balloon).

Para estar seguros de su correcto comportamiento, probamos el mismo funcionamiento con otro objeto, un cubo de basura. Con la ayuda de Rviz obtenemos visualmente la salida de los tópicos, ya sea de cámara o de imagen, de nuestro robot Pepper.

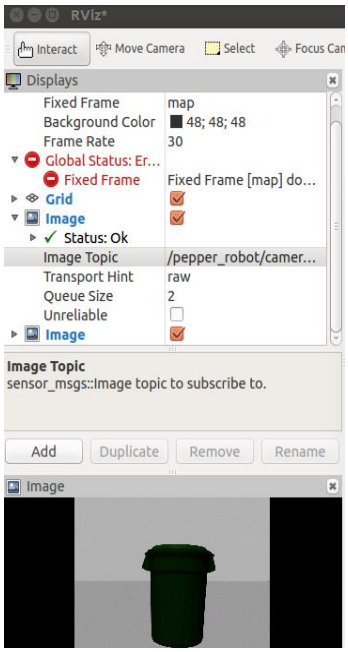


Figura 17 - Visión Cubo basura en RVIZ

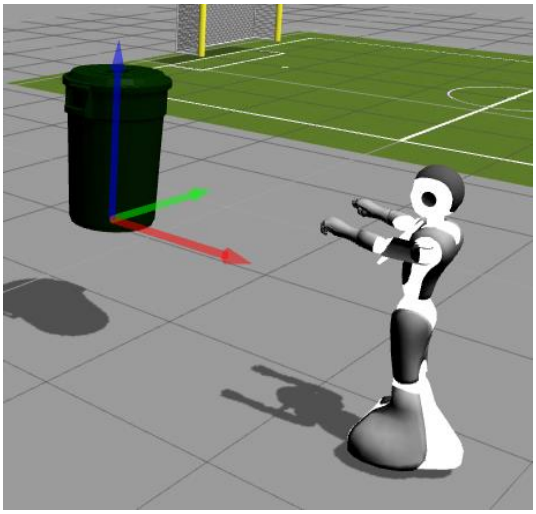


Figura 16 - Gazebo Pepper frente a cubo

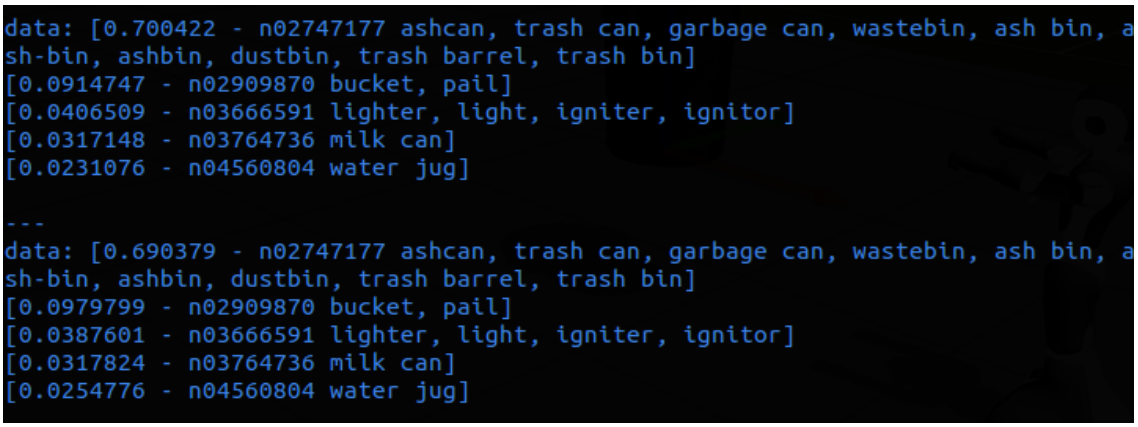


Figura 18 - Resultado ROS-Caffe con Cubo basura

Como se ha visto anteriormente, la primera de las aproximaciones es mucho mayor y cercana al 1. Por el contrario las otras están cerca del 0, por lo que podemos decir, que este objeto es un trash can (cubo de basura).

4.8.1 Sobre el código:

Ros init, inicializa ROS especificando el nombre del nodo (Cada nombre tiene que ser único). Se debe añadir en la parte superior del código.

```
ros::init(argc, argv, "tfg");
```

Node handle, crea un "apoyo" para este nodo. El primer handle creado inicializa el nodo y el último lo destruye.

```
ros::NodeHandle nh;
```

Respecto a los tópicos, vemos como se crea un publicador/suscriptor en el mismo archivo tfg.cpp. Primero el nodo se suscribe al tópico "pepper_robot/camera/front/image_raw" el cual lo envía el simulador Gazebo (Con una cola de imágenes de 1). Una vez se recibe esa imagen en el tópico,

ejecuta el método “imageCallBack”, el cual extrae la imagen del formato recibido y ejecuta el método “publishRet” con los resultados. Este método envía la información al tópico (publicador) que se crea en el objeto “gPublisher” (Tópico “/caffe_ret”).

```
const std::string RECEIVE_IMG_TOPIC_NAME = "pepper_robot/camera/front/image_raw";
const std::string PUBLISH_RET_TOPIC_NAME = "/caffe_ret";
```

```
image_transport::Subscriber sub = it.subscribe(RECEIVE_IMG_TOPIC_NAME, 1, imageCallback);
gPublisher = nh.advertise<std_msgs::String>(PUBLISH_RET_TOPIC_NAME, 100);
```

```
void imageCallback(const sensor_msgs::ImageConstPtr& msg) {
    try {
        cv_bridge::CvImagePtr cv_ptr = cv_bridge::toCvCopy(msg, "bgr8");

        cv::Mat img = cv_ptr->image;
        std::vector<Prediction> predictions = classifier->Classify(img);
        publishRet(predictions);
    } catch (cv_bridge::Exception& e) {
        ROS_ERROR("Could not convert from '%s' to 'bgr8'.", msg->encoding.c_str());
    }
}


void publishRet(const std::vector<Prediction>& predictions)  {
    std_msgs::String msg;
    std::stringstream ss;
    for (size_t i = 0; i < predictions.size(); ++i) {
        Prediction p = predictions[i];
        ss << "[" << p.second << " - " << p.first << "]" << std::endl;
    }
    msg.data = ss.str();
    gPublisher.publish(msg);
}
```

Figura 19 - Código tfg.cpp

4. 9 FAST R-CNN

Como ya hemos comentado anteriormente, la r-cnn es un framework de detección de objetos basado en redes convolucionales profundas, las cuales incluyen una Region Proposal Network (RPN) y una red de detección de objetos. Esta estructura le permite filtrar la imagen de entrada y utilizando ciertas características en la imagen de salida (valores numéricos elevados) se

puede llegar a obtener la zona de la imagen donde puede estar alojado un objeto. Hemos utilizado fast R-CNN basado en python.

En nuestro caso utilizamos solo la CPU ya que caffe es incapaz de conectarse a la tarjeta gráfica por un fallo en los drivers de la misma. Es aconsejable, si es posible, utilizar la GPU para trabajos de tanta carga computacional.

Para empezar la instalación, se debe clonar el repositorio recursivamente para asegurarnos que las subcarpetas y sus archivos se copien adecuadamente.

```
$ git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
```

Uno de los requisitos más importantes para la instalación es Caffe. Debe encontrarse instalado y configurado en la máquina en la que se pretende instalar Faster R-Cnn. Esta instalación de caffe suele llevar muchos problemas de compilación por falta de librerías (Fallo en **protobuf** [2])

Una vez estamos seguros de que Caffe está instalado, solo falta realizar el último paso.


```
$ git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
```


A continuación mostramos una demo sobre su funcionamiento mediante **CPU** (Ya que los drivers de Nvidia para el uso de la GPU generan errores).

```
tonda@TondaPC:~/Escritorio/FRCN_ROOT$ ./tools/demo.py --cpu
```

Figura 20 - Demo Faster R-cnn Perro

Figura 21 - Demo Faster R-cnn Persona

Como podemos observar, la red es capaz de detectar varios objetos en una misma imagen, aislando la zona donde posiblemente se encuentre dicho objeto. Esto nos permite obtener las coordenadas del mismo.

Una de las posibles aplicaciones de Faster R-Cnn es la utilización de dicha red neuronal dentro de ROS. En concreto, adaptándolo para que pueda utilizarse en el paquete ROS_Caffe. Esto permitiría una "mejora" en los resultados obtenidos, pudiendo así, obtener las coordenadas del objeto detectado que puede ser útil para realizar otros proyectos como "Seguimiento de objetos".

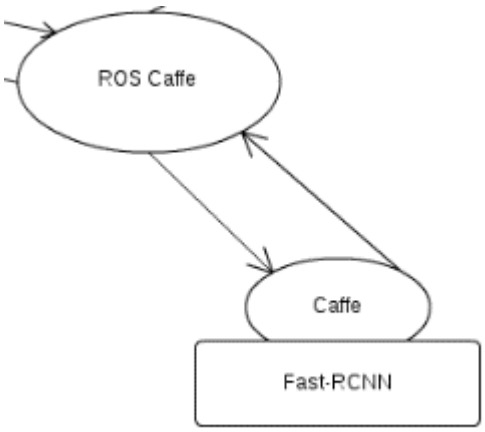


Figura 22 - Esquema con Faster R-cnn - Caffe

Los resultados de Fast-RCNN, son un pack de coordenadas x1, x2, y1 e y2. Estás coordenadas son enviadas a un nodo que calcularía la media de dichas coordenadas para encontrar el punto medio donde se encuentra el objeto detectado. Esta coordenada media sería enviada al tópico de movimiento de la cabeza del robot para así centrar la cámara con el objeto (realizándolo cada vez que recibiera coordenadas).

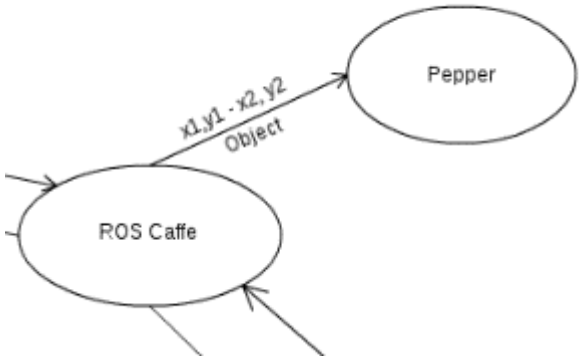


Figura 23 - Esquema con Faster R-cnn - Pepper

5. Conclusión

Tras la realización de esta guía se puede decir que se han cumplido todos los objetivos marcados desde el principio del mismo. Se ha conseguido instalar el software Ros Jade configurarlo de manera adecuada para conseguir ejecutar el simulador Gazebo y conseguir así una virtualización del robot Pepper. ROS nos ha permitido interactuar con el robot mediante los paquetes de Catkin.

Otra meta alcanzada ha sido conseguir que el robot Pepper distinga objetos a tiempo real según las imágenes que recibe en cualquiera de sus cámaras. Esto se ha realizado gracias al paquete ROS_Caffe, dentro de Catkin, y la adaptación de su código para obtener las imágenes del robot y procesarlas.

Por último se ha desarrollado la red neuronal Faster R-cnn para poder abrir la puerta a nuevos proyectos basados en el seguimiento de objetos.

Completados estos tres objetivos podemos decir que se ha completado el objetivo principal del cual derivan, crear una guía para el manejo de sistemas basada en ROS, Pepper y deep learning.

Con todo esto, podemos decir que se han tratado todos los aspectos básicos de estas tecnologías y que una persona con conocimientos básicos de la informática, después de hacer hincapié en esta guía, podría ser capaz de crear su propio proyecto deep learning con el robot Pepper.

Para concluir, y como mejora, se podría proponer como siguiente objetivo que la red Faster R-Cnn se adaptara al sistema de paquetes de ROS para conectarla y utilizarla con el robot Pepper.

6. Bibliografía y Referencias

1. http://wiki.ros.org/ROS/Tutorials – 2017

2. https://github.com/google/protobuf/tree/master/src – 2017

3. https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/ - 2017

4. http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf – 2017

5. http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf – 2017

6. https://github.com/rbgirshick/py-faster-rcnn – 2017

7. https://es.wikipedia.org/wiki/Red_neuronal_artificial – 2017

8. https://github.com/ros-naoqi/pepper_virtual/tree/master/pepper_gazebo_plugin – 2017

9. https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico – 2017

10. https://www.ald.softbankrobotics.com/en/robots/pepper/find-out-more-about-pepper – 2017

11. http://aliverobots.com/robot-pepper/?gclid=Cj0KEQjw9r7JBRCj37PlltTskaMBEiQAKTzTfDZ58cUaF2gSOuQYt7rN2BF4QPRj90CBuR4dYEimlu4aAgh18P8HAQ – 2017

12. https://github.com/BVLC/caffe – 2017

13. http://caffe.berkeleyvision.org/installation.html – 2017